D-10/01/20

Q1 Input 3 no using scanner class & determine a+b=c , a*b=c & a=b+c are true or not.

Q2 Find out if a no. is odd with out using multiplication, / & %.

```
Public static void main(String args[])
{
    Scanner z=new Scanner(System.in)
    int a,b,c;
    sopl("Input 3 no");
     a=z.nextInt();
     b=z.nextInt();
     c=z.nextInt();

  if(a+b==c)
    sopl("true");
  elseif(a*b==c)
     sopl("true");
  elseif(a==b+c)
     sopl("true");
  else
       sopl("false");
}
```

Q2.

```
psvm()
{
  int x;
   x=z.nextInt();
   odd(x);
}
```

```
public static void odd (int x)
{
    while ( x>0)
        x = x-2;
    if (x==0)  sopl("even");
    else  sopl("odd");
}
```

Q. find out the list of words generated from the following characters as 'y', 'e', 'a', 'r', if the characters are considered only once.

```
PSVM()
{
    int i, j, k, l, c=0;
    ~~String x = "year";~~   char x[] = {'y','e','a','r'};
    for( i=0; i<x.length; i++)
    {
        for(j=0; j<x.length; j++)
        {
            for(k=0; k<x.length; k++)
            {
                for(l=0; l<x.length; l++)
                {
    if (i!=j) && (i!=k) && (i!=l) && (j!=k)
                && (j!=l) && (k!=l))
                {
                    String y = x[i] + x[j] + x[k] + x[l];
                    solpn(y);
                }
            }
        }
    }
}
```

Q. Input a no. i.e greater than 2 & print the no. of times one must divide the no. by 2 before getting a value less than 2.

```
psvm( )
{
int n;
n = sc.nextInt( );
int c = 0;
if (x<=2) sopl("Enter a no. >2");
else
while (n > 2)
{
n = n/2;
c = c++;
}
sopln(c);
}
```

Q. Input 2 arrays & findout their dot product.

```
psvm( )
{
scanner z = new scanner(system.in);
int i[], j[], k[]; //scanner class
i[]=
for (int a=0; a<i.length; a++)
k[a] = i[a] * j[a];
for (int a=0; a<k.length; a++)
sopl(k[a]);
}
```

ID-11/01/20

CLASS

⟨access specifier⟩ class ⟨class name⟩

```
{
    member attributes;
    member methods();
}
PSVM()
{
    classname obj = new constructor();
    obj.attribute;
    obj.method();
}
}
```

class - run time entity

Q. Create a class student with the attribute name, regd. no, branch & methods input & disp. Input & disp. 3 students details.

file name - student.java.

```
public class student
{
    String name, branch;
    int regd;
    void input()
    {
     Scanner sc = new Scanner();
     name = sc.next();
     branch = sc.next();
     regd = se.nextInt();
    }
    void disp()
     { sopln(name + regd + branch);
    }
```

```
psvm ( )
{
  Student s1 = new Student ( );
   s1 . input ( );
   s1 . disp ( );
  Student s2 = new Student ( );
   s2 . input ( );
   s2 . disp ( );
  Student s3 = new Student ( );
   s3 . input ( );
   s3 . disp ( );
}}
```

Create multiple classes

Q Define another class abc that will store the student class obj. & access the members

filename - Abc . java

```
class Student
{
  String name , branch;
    int regd;
   void input ( )
   {
     Scanner sc = new Scanner (.-);
     name = sc . next ( );
     branch = sc . next ( );
     regd = sc . nextInt ( );
   }
   void disp ( )
   { sopln ( name + regd + branch );
   }}
```

```
public class Abc
{
    psvm()
    { student s1=new student();
      s1.input();
      s1.disp();
    }
}
```

Constructor:

→ It is a method used for creating obj. of a class.

→ Its name is same as class name.

→ constructors can be used for inputing purpose.

```
class student
{
  String name;
  int regd;
  student()  → constructor (default)
  {
    name = "xyz";
    regd = 1;
  }
  void disp()
  { sopln(name + regd);
  }
}
    public class Abc
    { psvm()
      { student s1 = new student();
        s1.disp();
      }}
```

Parameterised Constructor :

When arguments are passed in a constructor it is called as parameterised constructor.

→ If a prog. contains a parameterised constructor, the default constructor must be defined otherwise the prog. will give error.

```
class student
{
    String name, branch;
    int regd;
    student()
    {
        name = "xyz";
        branch = "cse";
        regd = 1;
    }
Student(String n, String b, int r)
    {
        name = n;
        branch = b;
        regd = r;
    }
    void display()
    {
        sopln(name+branch+regd);
    }
}
public class Abc
{
    psvm()
    {
        student S1 = new student();
```

```
        s1.display();
        Student s2 = new student("pqr", "CSE", 2);
    }
}
```

D-17/01/20

Static Variable

If a variable is defined as static it will be commonly available to all the objects of the class

```
Class student
{
 String name;
 int roll;
 static String college = "ITER";
 student()
{
}
student (String n, int r)
 { name = n;
   roll = r;
 }
 void disp()
  { sopln (name + college + roll);
  }
 }
 public class xyz
    { psvm()
    { student s1 = new student("A", 1);
      student s2 = new student("B", 2);
      s2.disp();
    }
}
```

Static method.

If a method is defined as static the it can only access the static variables

```
class stud
{ string name;
  static string b="CSE";
  static string college="ITER";
  Student()
  {  }
  stud (string n, int r)
  { name=n;
    roll=r;
  }
  static void x()
  { b="csit";
  }
  .
  .
  .
  psvm()
  { stud.x();
    :
  }
```

If a prog. contain a parmeterised constructor it must include a default constructor, other wise it will show an error.

```
class Abc
{ int x = 1;
  Abc()
  { x++;
    Sopln(x);
  }
}
public class B
  { psvm()
    { Abc P = new A();
      Abc q = new A();
    }
  }
```

O/P 2.
2

the compiler will intialized x=1 for each & every obj.

*

```
class A
{
static int x=1;
  A()
  { x++;
    Sopln(x);
  }
}
  public class B
  { psvm()
    { A P = new A();
      A P = ...
```

O/P 2
3

If x is defined as static then the compiler will consider the latest value of x.

Static block

If a static block is present it will execute before the main method.

Package-

Package is a collection of diff classes.

Import key word is used for using a package.

Access specifier

It is used for providing security to a prog.

(i) Public
(ii) Private
(iii) Protected
(iv) no-access (default)

	Inside class	Outside class	Inside package	outside package
Public	yes	yes	yes	yes
private	Yes	NO	NO	NO
protected	yes	Yes (through inheritance)	Yes (through inheritance)	~~Yes~~ (through NO inheritance)
default	Yes	~~NO~~ yes	Yes	NO

D-18/01/20

```
class A
{
int x=10;
protected int y = 20;
public float z = 30;
private double p=40;

void m1( )
{
Sopln (x+y+z+p);
}

private void m2( )
{
Sopln(x+y+z+p);
}

public void m3( )
{ Sopln( x+y+z+p);
}
protected void m4( )
{ Sopln (x+y+z+p);
}
}
public class B
{ psvm( )
{
A m=new A( );
m.m1( );
m.m2( );
m.m3( );
m.m4( );
```

```
sopln(m.x); -
sopln(m.y); -
sopln(m.z); -
m.p X
```

Inheritance -

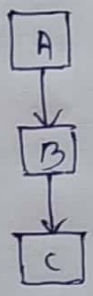
A child class is inherited from the parent class.

(i) Single
(ii) Multilevel
(iii) Multiple
(iv) Hierarchical
(v) Hybrid

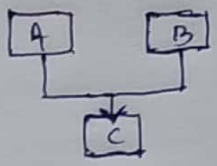
* Java doesn't support multiple inheritance.

(i) Single

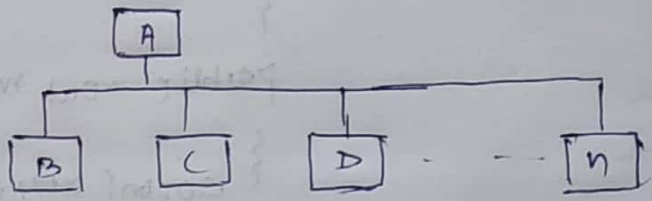
(ii) Multilevel



(iii) Multiple



(iv) Hierarchical



(v) Hybrid

Comb^n of any 4.

Single

→ The parent is otherwise called as super or base.

→ The child is " " " derived or inherited.

→ For inheritance extends keyword is used.

child extends parent.

Single

```
class A
{
  int x=10;
}
class B extends A
{
  int y=20;
}
```

```
public class X
{ psvm()
  {
    B P=new B();
    Sopln(P.x+P.y);
  }
}
```

O/P 30

→ The child class can always access the parent class & its members

→ So in main fun obj. of child class is defined

★

```
class A
{ A()
  { sopln("a");
  }
}
class B extends A
  { B()
    { sopln("b");
    }
  }
```

```
public class X
{
  psvm()
  { B P = new B();
  }
}
```

O/P a
b

D-22/01/20 If a constructor defined as private it can't create object outside the class

```
public class X
{
  int P=10;
  Psvm()
  { sopln(P);
  }
}
```

→ Error

→ Static main method can't access non static var. P.

Multilevel

```
Class A
{
  A()
  {
    sopln("a");
  }
}
Class B extends A
{
  B()
  { sopln("b");
  }
}
```

```
class C extends B
{
  C()
  { sopln("c");
  }
}
public class X
{ psvm()
  { C p=new C();
  }
}
```

super

It is used to refer the immidiate parent class members.
It is automatically used, we dont hv to write.

```
class A
{ int a=20;
  A()
  { sopln(a);
  }
}
class B extends A
{ int a=30;
  B()
  { super();
    sopln(a);
  }
}
```

```
public class X
{
  PSVM()
  { B p=new B();
  }
}
```

O/P 20
30

```
class A
{
  int a=20;
  {
    class B extends A
    { int a=30;
      void disp()
      { sopln(a);
      }
    }
public class X
{ psvm()
  { B p= new B();
    p.display();
  }
}
```

O/P = 30

```
class A
{ int a=20;
  { class B extends A
    { int a=30;
      void displ()
      { sopln(a);
        sopln(super.a);
      }
    }
```

```
public class X
{
  psvm()
  {
    B p = new B();
    p.disp();
  }
}
```

O/P 30
20

*

```
class A
{
  int a = 20;
  void disp()
  {
    sopln(a);
  }
}
class B extends A
{
  int a = 30;
  void disp()
  {
    sopln(a);
  }
}
public class X
{
  psvm()
  {
    B p = new B();
    p.disp();
  }
}
```

O/P. 30

* If the base class defined as private then the ~~child~~ child class can't access any member of the base class.

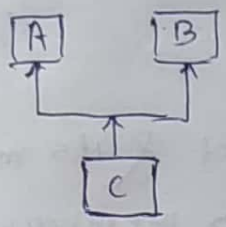
Base class	derived class		
	private	public	protected
private	No	No	No
protected	private	protected	protected
public	~~yes~~ private	~~yes~~ public	~~yes~~ protected

*

```
class A
{
  int a=20;
  void disp()
  { sopln(a);
  }
}
class B extends A
{ int a=30;
  void disp()
  { sopln(a);
    super.disp();
  }
}
public class X
{
  psvm()
  { B p=new B();
    p.disp();
  }
}
```

O/P 30
20

Multiple




```
class A
{ int a=20;
  void disp()
  { sopln(a);
  }
}
class B
{ int a=30;
  void disp()
  { sopln(a);
  }
}
class C extends B, A
  { int a=40;
  }
public class X
  { psvm()
    { C p=new C();
      p.disp();
    }
  }
```

→ error / confusion / ambiguity

In this prog the compiler gets confused to use which disp fun^n. this creates a ambiguity which can be resolved by

an concept called as interface in java.

This pointer

→ Only refers to the current class object & its members

→ This is used to distinguish betn the class members & local members if same names are used for both,

```
void setdata(string name, int roll, double mark)
{
    this.name = name;
    this.roll = roll;
    this.mark = mark;
```

* Package is a collection of diff. classes.

```
package P1;
public class A
{
    public int x=10;
}
```

→ default/private/protected - cant be accessible in package P2 error

```
package P2;
import P1*;
public class B
{
    psvm(string[] args)
    {
        A P=new A();
        sopl(P.x);
    }
}
```

D-24/01/20

```
private class A
{
  protected A()
  {
    sopln("A");
  }
}
class B extends A
  {
    B()
    { sopln("B");
    }
  }
```

O/P error (bcz private)

Or ②

```
public class C
{psvm()
 { A x=new A();
   x.show();
   A y=new B();
   y.show();
 }
}
```

O/P parent child

```
public class C
  { psvm()
    { B x=new B();
    }
  }
```

Method over-riding
Same method name & same signature

The object x in ② is created by referring to class A.
~~show disp~~ x.show will point to parent.

The object y in ② is created by referring to class B.
Thats why y.show points to the child.

→ The compiler decides which show function to execute during the running of the program.

→ This is called as runtime polymorphism.

↓
same fun^n name diff forms.

★ A fun^n will act in many diff ways is called polymorphism.

```
class A
{ void show()
  {sopln("parent");
  }
}
class B extends A
{ void show()
  {
    sopln("child");
  }
public class c
{ p.s.v.m()
  {
    B x = new B();
    x.show();
    B y = new A();
    y.show();
  }
}
```

O/P
error

Object of child (B) can't be created by ref. to the parent class constructor.

```
public class c
{
  ps
```

* If the show() of cls A is protected then it will show error bcz the compiler is trying to assign weaker access specifier to show fun^n.

```
class A
{
  protected void show()
  {sopln("parent");
  }
}
```

be

```
class B extends A
{
  public void show()
  { sopln("child");
  }
}
public class C
  {
    psvm()
    {
      B x = new B();
      x.show();
      A y = new BC();
      y.show();
    }
  }
```

D-25/01/20

Q. Complex → class

real : variable (int)

imag : variable (int)

display() : method (void)

add (complex, complex) : method

use constructor for input

Ans.

```
class complex
{
  int real, imag;
  complex(int r, int i)
  { real=r;
    imag=i;
  }
  void add (complex c1, complex c2)
  { real = c1.real + c2.real;
    imag = c1.imag + c2.imag;
  }
```

```
    void disp()
    { soptln (real + "+i" + imag);
    }
}
    public class demo
    {
        psvm()
        { complex c1 = new complex (1,2);
          complex c2 = new complex (3,4);
          c2.add(c1, c2);
          c2.disp();
        }
    }
```

Q. How to return object in a method ?

```
class complex
{ int real, imag;
  complex(int r, int i)
   { real = r;
     imag = i;
   }
  complex add (complex c1, complex c2)
   { complex c3 = new complex ();
     c3.real = c1.real + c2.real;
     c3.imag = c1.imag + c2.imag;
     return c3;
   }
  void disp()
   { soptln(real + "+i" + imag);
   }
```

```
public class Demo
{
    psvm()
    {
        complex x = new complex(1, 2);
        complex y = new complex(3, 4);
        complex z = new complex();
        z = z.add(x, y);
        z.disp();
    }
}
```

Q. Create a class student that contains name, roll & course as var. & input & disp as methods. Another class exam is derived from the student. Exam contains m1, m2, m3 as variables of 3 diff. subjects and input-marks() & disp-total() as methods. Create an array of objects of exam to display 10 student detail & their marks.

```
class student
{ string name, course;
  int roll;
 void input (string name, string course,
                                int roll)
  {
    this.name = name;
    this.course = course;
    this.roll = roll;
  }
   void disp()
  { sopln(name + roll + course);
  }
}
```

```
class Exam extends student
{ int m1, m2, m3;
  void input_mark(int m1, int m2, int m3)
  { m1 = x;
    m2 = y;
    m3 = z;
  }
  void display_total()
  { sopln(m1+m2+m3);
  }
}
public class demo
{ psvm()
  {
    Exam e[] = new Exam[10];
```

(This line means an array of size 10 has been created)

```
for(i=0; i<10; i++)
    e[i] = new exam();
```

(object creation)

```
Scanner sc = new Scanner(System.in);
  for(i=0; i<10; i++)
  { e[i].input(sc.next(), sc.next(),
                    sc.nextInt());
    e[i].input_mark(sc.nextInt(), sc.nextInt(),
                    sc.nextInt());
  }
```

```
for(i=0; i<10; i++)
  {
    e[i].disp();
    e[i].display-total();
  }
}
```

ID-29/01/20

```
class Person
{
  String name;
  int age;
  Person(String n, int a)
    {
      name=n;
      age=a;
    }
  void display()
  {
    S.O.P("Name: "+name+"Age: "+age);
  }
}
class Student extends Person
  {
    super(n,a);
    String course;
    int roll, marks;
    Student(String c, int i, int m)
    {
      course=c;
      roll=i;
      mark=m;
    }
    void display()
    {
      super.display();
      SOP("Course: "+course+"Roll: "+
          roll+"Mark: "+mark);
    }}
```

```
class Teacher extends Person
{
    String sub_assign;
    int hour;
    Teacher(String s, int h)
    {
        super(n, a);
        sub_assign = s;
        hour = h;
    }
    void display()
    { super.display();
      SOP("Sub: " + sub_assign + "Hour: " + hour);
    }
}
```

Main

```
public class demosuper
{ public static void main(String args[])
  {
    Student s1 = new Student("Ipsita" + 18 +
                 "CSE" + 387 + 30);
    s1.display();
    Teacher t = new Teacher("Deepti" + 25 +
                "DSA" + 5);
    t.display();
  }
}
```

O/P Ipsita 18 CSE 387 30

Deepti 25 DSA 5

D-31/01/20

If a parent class contains a protected method & a child class is overriding it with a default method, the comp. will generate error.

→ But it can be over-ridden by a public method.

→ Static method of parent can-not be over-ridden by the child.

→ If both parent & child contain static method

```
class A
{
static void show()
  {
    sopln("A");
  }
}
class B extends A
 {
static void show()
  {
    sopln("B");
  }
 }
public class demo
  {
    psvm()
     {
       A  p=new A();
       P.show();
       A  q=new B();
       q.show();
     }
  }
```

O/P A
A

Use of final key-word in java

(a) Variable

→ If a var. is declared as final it becomes global

→ If a var. declared as static & final it becomes a const.

```
final static int a = 10;
```

Any operations working on const. will give errors

(b) Method

→ If a method is defined as final, it can't be over-ridden

(c) Class

→ If a class is defined as final, it can't be inherited

Interface-

→ It is a keyword used for availing multiple inheritance.

→ All the var. & methods present in interface are public in nature.

→ It doesn't contains any method body rather it contains only method signature.

→ The class implementing the interfaces have to define the method body

~~→ Imp~~

Implements keyword is an used for interfaces.

```
interfaces A
{ void show();
}
interface B
{ void show();
}
```

```
class C implements A, B
{
    void show()
    {
        sopln("P");
    }
}
public class Demo
{
    psvm()
    {
        C x = new C();
        x.show();
    }
}
```

D-05/02/20

Recursion

Method calling itself is called as recursion

Factorial-

```
class A
{
    psvm(String args[])
    {
        int x = 4, f;
        f = fact(x);
        sopln(f);
    }
    int fact(int n)
    {
        if(n==0)
            return 1;
        else return n * fact(n-1);
    }
}
```

execution pattern :-

```
                          n
main()  <--
  ⇓         |  24
fact(4)  ----      4
  ⇓
      24
4 * fact(3)        3
  ⇓
      6
3 * fact(2)        2
  ⇓
      2
2 * fact(1)        1
  ⇓
      1
1 x fact(0)        0
```

* A method will return a value to the exact location where it is being called.

* For writting recursion prog. 1st find out the repeatative stmt & stopping criteria

Reverse -

```
class A
{ static int r=0;
  psvm(..)
  { int x=412, f;
    f = rev(x);
    sopln(f);
  }
  int rev(int n)
  { r=0;
    if(n)
    { d= n%10;
      r=r*10+d;
      rev(n/10);
    }
    else return r;
  }
}
```

n	r	d	main()
~~412~~	~~0~~	~~2~~	↓ rev(n)
~~41~~	~~2~~ 21	~~1~~	
4	214	4	
0			

Sum of digits

```
class A
{
  psvm(String args [])
  {
    int x = 213, s;
    s = sum(x);
    sopln(s);
  }
  int sum(int n)
  {
    int a=0;
    if
    while
    if (n)
    {
      d = n % 10;
      a = a+d;
      sum (n/10);
    }
    else return a;
  }
}
```

Fibonacci series

```
int fib (int n)
{
  if (n<=1)
    return n;
  else
  return fib(n-1) + fib(n-2);
  }
  psvm()
  { fib(8);
  }
}
```

Abstract class

→ A class whose object can't be created by its own constructor is called abstract class.

→ Abstract keyword is used for making a class abstract

→ An abstract class can be extended using inheritance.

```
abstract class x
{
  int y=10;
}
class z extends x
 {
  int y=20;
 }
public class ABC
{
   psvm(String[] args)
   {   // x p=new x(); // error
     x p1=new z();
     sopln(p1.y);
     z q = new z();
     sopln(q.y);
   }
}
```

O/P 10
20

★ A abstract class can contain an abstract method & normal method.

Concrete class

★ A class whose obj can be created by its own constructor then it is called as concrete class

★ If a method is defined as abstract it has to be over-ridden by the child class.

★ Abstract class provides skeleton or base for other classes & methods to use.

Difference betn Abstract class & interface.

* A abstract class can contain both ~~abstr~~ abstract & normal method
→ By default all the methods are abstract in nature.
* Interface supports multiple inheritance.
→ Abstract class doesn't.

Interface:

```
interface t
{
  void f1();
}
class x11 implements t
{
  public void f1()
  {
    sopln("hi");
  }
}
public class abc
{
  public static void main(String args[])
  {
    x11 obj = new x11();
    obj.f1();
  }
}
```

In interface all the method should be public.

D-07/02/20

Find x^n by recursion

```
int power(int x, int n)
{
  if(n != 0)
  return x * power(n-1);
  else return 1;
}
```

palindrome by recursion

```
int rev(int n)
{
  r=0;
  if(n)
  {
   d = n%10;
   r = r*10+d;
   rev(n/10);
  }
  else
    return r;
}
boolean palindrome(int n, int p)
{
  if(n==p)
   return true;
  else
   return false;
}
```

```
class A
{ PSVM(String args[])
  { int x=412, t, q;
    t=rev(x);
    q=palindrome(x,t);
    sopln(q);
  }
}
```

D-08/02/20 Gcd of 2 no.s

```
int gcd(int p, int q)
{ if(q==0)
    return p;
  else
    return gcd(q, p%q);
}
```

String Reverse

```
String rev (String s)
{
 if( s.isempty() )
      return s;
 else
   return( rev(s.substring(1)) + s.charAt(0);
```

execution

```
      0 1 2
 s:   a b c
```

```
= rev(bc) + a
= rev(c) + b + a
= rev( ) + c + b + a
= "" + c + b + a = cba
```

palindrome

```
    main()
    {
      int x=434, s;
      s=palindrome(x);
      if( s==x)
       sopln("palindrome");
      else
        sopln("not");
    }
    int palindrome( int x )
    { if (x!=0)
      { d=x%10;
        r=r*10+d;
        palindrome(x/10);
      }
      else r
          return r;
       return r;
    }
```

String Palindrome

```
main()
{
  String str;
  if(palindrome(str))
    sopln("palindrome");
  else
    sopln("not");
  {
  boolean palindrome(String str)
  {
    if(str.length()==0 || str.length()==1)
      return true;
    if(str.charAt(0)==str.charAt(str.length()-1))
      return palindrome(str.substring(1,str.length()-1));
    return false;
  }
```

ID - 12/02/20

Tower Of Hanoi

The game contains 3 rods, n discs. Initially all the disc are on rod a & the the largest one in the bottom & smallest on at the top. the task is to move entire stalk of disc from a → c, without changing the structure.

Rules

1. Move only one disk at a time.
2. A disc can only be moved it it is at the top
3. Larger disc can't be placed at the top of a smaller one.
4. If n discs are there there will be $2^n - 1$ steps.

```
class A
{
  psvm(string args[])
  { int n=3;
    tower(n,"A","B","C");   → method
  }
public static void main tower(int n, String A,
                              String B, String C)
  {
    if(n==1)
     sopln("move disk from A to C");
    tower(n-1, A, C, B);
    sopln("move disk" + n + "from A to B");
    tower(n-1, C, B, A);
  }
}
```

Product of 2 no. without using 2 no.s

```
public static int product(int x, int y)
{ if(x<y)
    return product(x,y);
  else if(y==0)
      return 0;
  else return(x+ product(x,y-1));
}
```

Sort An array

```
void sort(int a[], int n)
{ if (n==1)
   return a;
  for(i=0; i<n-1; i++)
  { if (a[i] > a[i+1])
    { x = a[i];
      a[i] = a[i+1];
      a[i+1] = x;
    }
  }
   sort (a, n-1);
}
```

Binary search

```
int bs(int a[], int first, int last, int key)
{ if (first > last)
     return -1;
  int mid = (first + last)/2;
  if( a[mid] == key)
       return mid;
  else if(key < a[mid])
     return bs(a, first, mid-1, key);
  else
     return bs(a, mid+1, last, key);
```

decimal to binary

```
int dec_bin(int n)
{ if(n==0)
    return 0,
  else return (d%2 + 10 * dec_bin(n/2)),
}
```

Exception handling mechanisms-

Abnormal termination of prog., even if the syntax & logics are correct.

Try.

Try keyword is used to denote where the exception is present.

Throw

Throw keyword is used to generate the exception.

Catch

Catch block, catches the exception & rectifies it.

D-14/02/20

If exception is present in a program, then the execution flow for the rest of the codes stop.

→ For avoiding this exception handling mechanism is used by the help of 5 keywords that are. try, throw, catch, throws, finally.

Throws-

→ If a method generates exception the throws keyword is used.

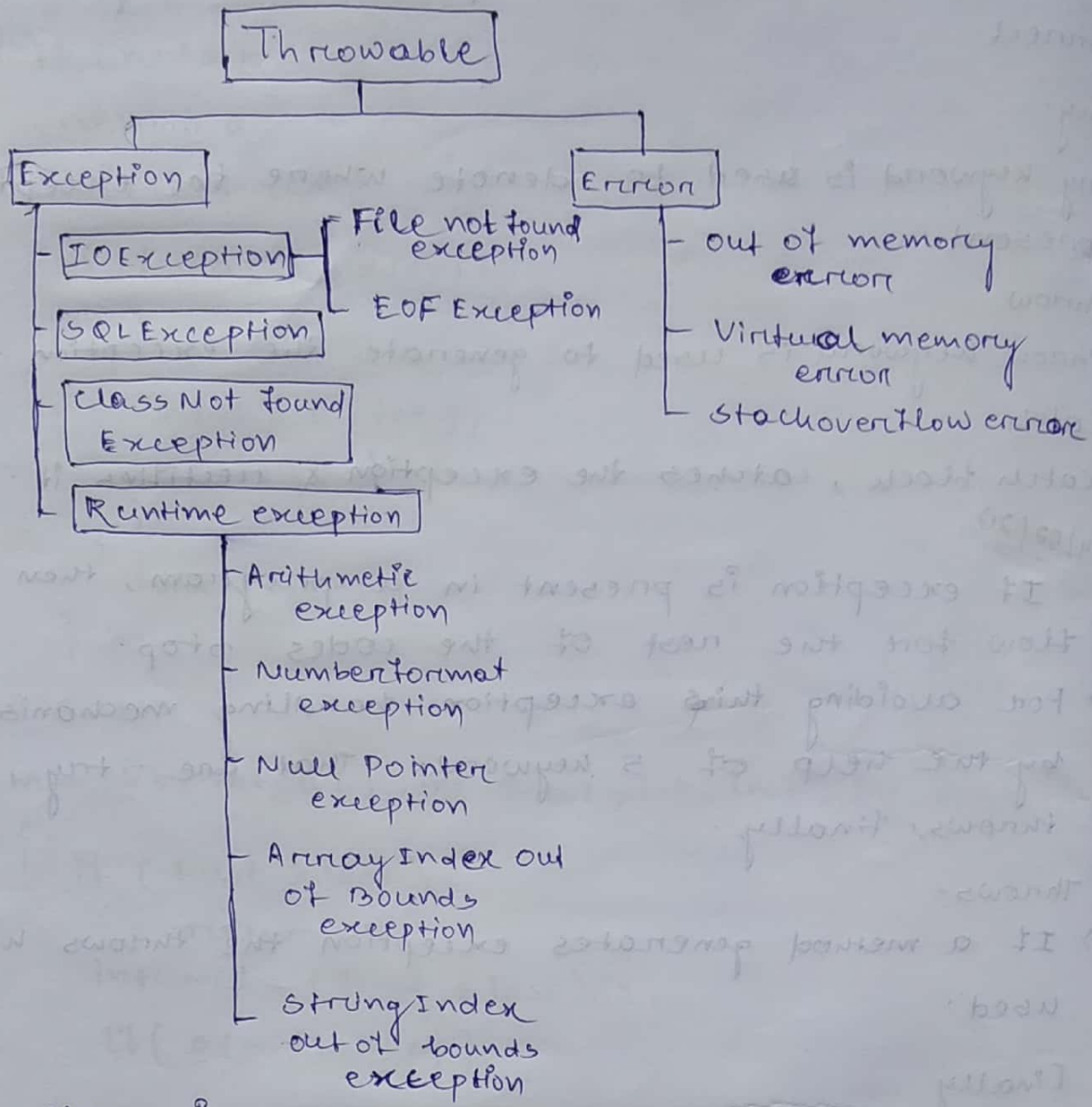
Finally

→ Its a block of statements executed even if there is no exception.

```
Psvm()
{ stmt 1;
  stmt 2; → exception
  stmt 3;
  stmt 4;
}
```

```
Psvm()
{ stmt 1;
  try
  { stmt 2;
    throw new exception();
  }
  catch (Exception object)
  { stmt 6;
  }
  stmt 3;
  stmt 4;
}
```

→ In java exception is a class, which is the child of throwable class



→ The error class is e-recoverable.

→ Exception is of 3 types (i) checked
(ii) Unchecked
(iii) Error

Checked

→ These are handled during compile time

→ Ex-All the exceptions except runtime exception

Unchecked

→ These are handled during runtime.

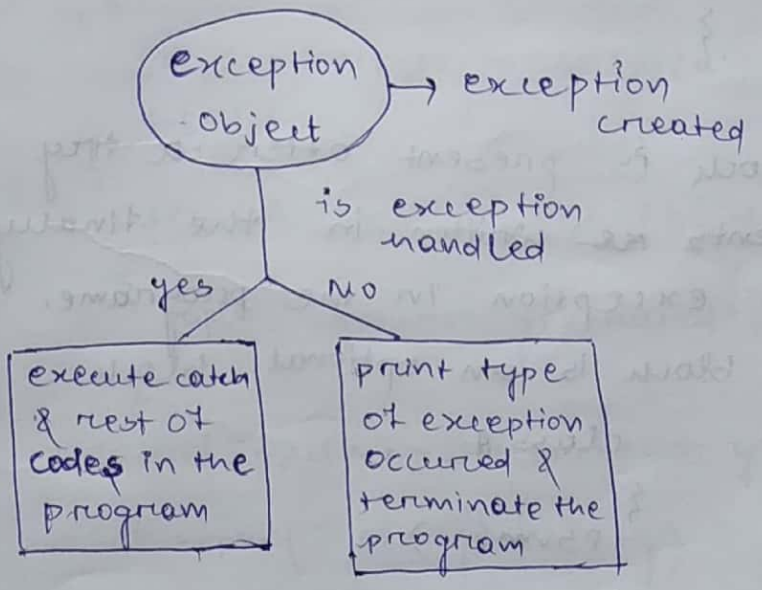
Ex-Runtime exception

→ When Exception is generated in the try block, the compiler executes the catch block & then the rest of the course will execute

```
Class A
{
  psvm( )
  {
    String a;
    int x=5, y=0, z;
    try
    {
      z = x/y;
      throw new Exception();
    }
    sopln(x+y);
    catch(Exception e)
    { sopln(e); sopln("E");
    }
  }
}
```

O/P = E 5

★



D-15/02/20 Q. Generate an error if an inputed no is a -ve no.

```
Class A
{
  ~~Rstm(System.out.println~~
  psvm(String args[])
  { int n;
    Scanner sc=new Scanner(System.in);
    n=sc.nextInt();
    try
    { if(n<0)
      throw new exception();
    }
    catch(Exception x)
    {sopln("Exc. generated");
    }

  }
}
```

Finally

-> A finally block is present after a try block.
-> The statements are written in the finally block even if there is no exception in the programe.
-> The finally block is an optional block.

```
class A
{
 psvm( )
 {
  Ent n;
  Scanner sc = new Scanner(System.in);
  n= sc.nextInt();
  try
  {
    if(n<0)
    throw new exception();
  }
```

```
catch (Exception x)
{ sopln("exception generated" + x);
}
finally
{ sopln(n);
}
}
}
```

A prog. can contain multiple try blocks & multiple catch block.

Multiple catch.

```
class A
{
  psvm(String args[])
  { String a = "xyz";
    int b[] = new int[4];
    try
    { int c = Integer.parseInt(a);
      b[5] = 72;
    }
    catch (ArrayIndexOutOfBoundsException x)
    { sopln("Exception generated" + x);
    }
    catch(InputMismatchException y)
    { sopln("exc. gene." + y);
    }
    finally
    { sopln("hello");
    }
  }
}
```

In this prog. only one exception will be generated, i.e Input mismatch. In this case the input mismatch exception is generate & the respective catch block will execute.

→ Even if the parent exception class is present, the control will always go to the child class exception.

```
class A
{
 psvm( )
 {
  String a="xyz";
   int b[]=new int[4];
  try
   {
    b[4] = 7/0;
   }
catch("Input miss match"y)
 {sopln("A" +y);
 }
catch(Exception z)
 { sopln(z);
 }
}
}
```

In this prog. the 7/0 will generate an exception. Since there is no appropriate catch block is present to it will handle the arithmatic exception, the control will go to the parent class exception.

Q create an array of string type and generate null pointer exception & array index out of bound exception.

```
class A
{
psvm(String args[])
{
 try
  String a[] = {"abc", null};
  try
   {sopln(a[1].length());
   }
  catch (Null pointerException x)
   {
    sopln("handled");
   }
```

```
try
{ a[3] = "xyz";
}
 catch (ArrayIndexOutOfBoundsException y)
  {soptln(y);
  }
}
```

D-19/02/20 Throws.

When a fun creates exception the throws keyword is used.

→ If never ~~throws~~ generate an exception.

→ when a fun throws exception the through keyword is used for generating the exception

```
class A
{
void show() throws Exception
 { throw new exception();
 }
 psvm( )
{ A abj = new A();
 try
   {obj.show();
   }
catch(exception P)
  { soptln(P),
  }
{
}
```

Q Difference bet^n throw & throws.

→ Throw is used to generate exception.

→ Throws never generate exception

→ Throw key word is used to create obj. of the exception class.

⇒ Throws is used to notify the compiler that an exception is going to be created by a method or a fun.

Q. How to create custom exception.
(user define)

By extending the exception class the compiler allows to create a custom or user define exception.

Q. Create a class student Enter name & mark. If mark is more than hundred create exception "MarksOutofBoundException" & throw it.

```
class student
{
  String name;
  int mark;
  Student (String n, int m)
  { name = n;
    mark = m;
  }

public class MarksOutofBoundException extends Exception
{
  psvm (String args[])
  {
    sopln("Enter name & mark");
    Scanner z = new Scanner(System.in);
    String name = z.next();
    int mark = z.nextInt();
    Student P = new Student (name, mark);
    try
    {
      if (P.mark > 100)
      throw new MarksOutofBoundException();
    }
    catch (MarksOutofBoundException e)
    { sopln("Exception handeled");
    }
```

```
        sopln(P.name + "    " + P.mark);
    }
}
```

ID-22/02/20

Generics

By using generics the user can define a single type for any datatype.

→ The representation of generics is <Type>

→ A class & a method can be of generic type.

→ If a class is generic type, it can take any type of class

∴ Ex- Integer
Double
Float
Object

→ If a method is generic it can return or take any data type as argument or variable

⇒ How to write generic method ?

Q. Create a class demo that contains a display() of generic type. Display the array elements of integer & double.

Without using generic

```
Psvm()
{ int a[] = {1, 2, 3};
  String b[] = {"a", "b"};
  double c[] = {1.1, 2.3, 3.3};
  for(i=0; i< a.length; i++)
    {
      sopln(a[i]);
      sopln(b[i]);
      sopln(c[i]);
    }
}
```

```
class Demo
{
  static <E> void display(E[] array)
  {
    for(i=0; i< array.length; i++)
      sopln(array[i]);
  }
  psvm()
  {
    Integer a[] = {1, 2, 3, 4};
    Double b[] = {1.1, 2.2};
    String c[] = {"a"};
    display(a);
    display(b);
    display(c);
  }
}
```

Q. Create a method count as generic type which will compute the no. of times an element is present in an array.

```
class X
{
  static <T> int count(T[] array, T item)
  { int c=0;
    for(T i: array)  [means for(i=0; i<array.length; i++]
    {
      if(i.equals(item))
        ct+;
    {
      return c;
    }
    psvm()
    { Integer a[] = {1, 2, 1, 3},
      Double b[] = {7.5, 6.5, 4.3, 7.5};
      sopln(count(a,1));
      sopln(count(b,4.3));
    }
}
```

O/p :
2
1

D-26/02/20

```
PSVM()
{ int m=9, n=6;
try
{
 while(m%n>=0)
 { sopln(m+" "+n);
   m=m+1;
   n=n-1;
 }
throw new ArithmeticException();
}
catch(ArithmeticException x)
 { sopln("exception handled");
 }
}
```

Q. Binary search using recursion

Recursive trace / Recursive tree

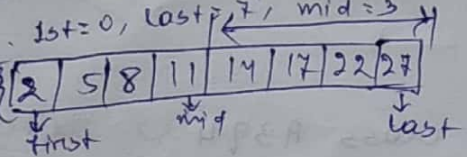
0	1	2	3	4	5	6	7
2	5	8	11	14	17	22	27

key = 14

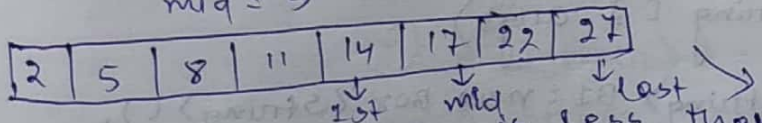
```
if key == a[mid]
   return mid;
else if (key < a[mid])
   bs(a, first, mid-1, key);
else
   bs(a, mid+1, last, key);
```

code

Since 14 is greater than mid, search the element in the ~~mid~~ mid+1 to last.

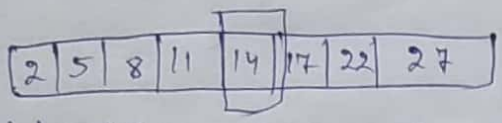


first = 4
last = 7
mid = 5



Since the key value is less than the mid search for the element in first to mid-1

first = 4
last = 4
mid = 4



Since mid = key, element is found at position 4.

Similarities betn interface & abstract class

→ Both contain abstract method.
→ The abstract method has to be overridden by the implemented or child class.
→ We can't create obj. of abstract cls & interface by their own ~~con~~ constructor.

Q1 Find greater betn 2 no. of any type using generics.

Q2 Create a class DOB that contains DD-MM-YY & set DOB method. Enter dob of 2 person & find yonger betn them.

Generic class :-

ABQ4

```
Class Box<T>
{
  T i;
  void input (T a)
  {  i=a;
  }
  void output()
  {
    sopln(i);
  }
}
public class A3Q4
{
  psvm(String [] args)
  {  Box<String> B1 = new Box<String>();
```

```
B1.input("Hello");
B1.output();
Box<Integer> B2 = new Box<Integer>();
B2.input(10);
B2.output();
Box<Object> B3 = new Box<Object>();
B3.input("ITER"); B3.output();
B3.input(29); B3.output();
}
}
```

D-28/02/20 ~~generic ex.~~

Q. soln for comparing any 2 types of elements using generics compareTo method is used which is present in the comparable class.

```
class A<X extends comparable>
{
 X a, b;
 A(X a, X b)
 {
  this.a = a;
  this.b = b;
 }
 void display()
 {
  if(a.compareTo(b) > 0)
   sopln("a is bigger");
  else
   sopln("b is bigger");
 }
}
public class demo
{
 psvm()
 {
  A<Integer> p = new A<Integer>(10,20);
  A<String> q = new A<String>("hello","Hello");
  p.display();
  q.display(); } }
```

Q Find the sum of elements in a array using recursion.

The problem can be solved using binary sum. that will use 2 binary sum recursive fun.

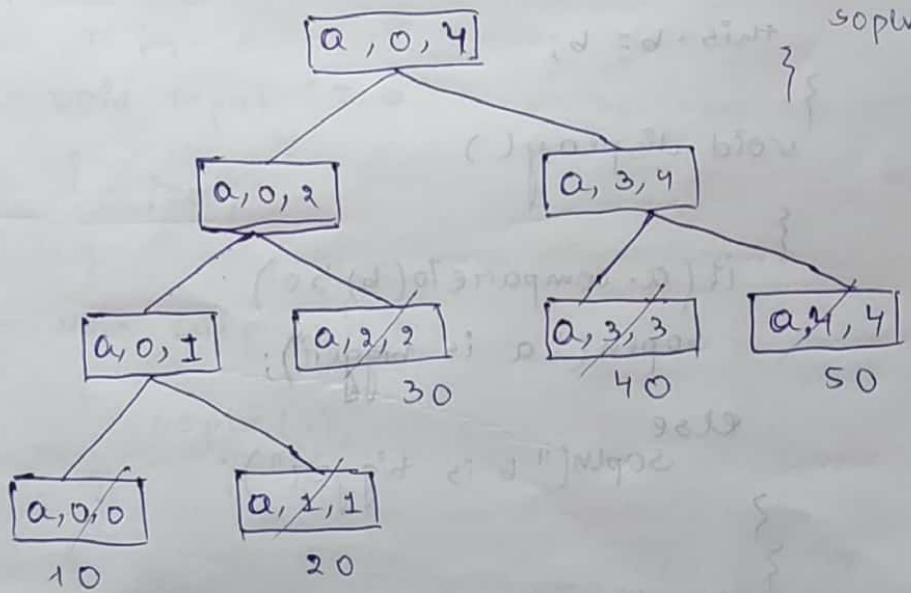
```
int binarysum ( int a[ ], int first , int end)
{
  if ( first > end)
    return 0;  // empty array
  else if ( first == end)
    return a [first ];  // single element
  else
    { int mid = (first + end) / 2 ;
      return binarysum ( a, first , mid) +
             binarysum ( a, mid + 1 , end );
    }
}
```

```
psvm ( )
{
  int a[ ] = {10, 20, 30, 40, 50}
  first = a[0]; end = a.length - 1
  int sum = binarysum(a, first, end)
  sopln (sum);
}
```

Tree

a[ ] =

10	20	30	40	50
0	1	2	3	4



D-11/03/20

DATA STRUCTURE

→ Data is the raw format of any input. It doesn't provide any meaning.

Information

→ Data represented in sentence which produce a meaning is called as information.

Data structure

DS is the way of representing the data to extract information.

→ It is a way of collecting & organising & manipulating the data in such a way that various operations can be performed on the data in an effective way.

Examples of ds

- Linked list
- stack
- queue
- tree
- graph
- array
- file

Linked list

It is a list consist of various items.

Stack

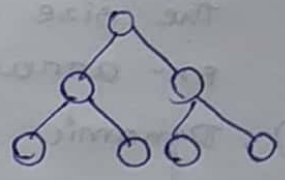
Collection of diff. elements in LIFO (last in first out) order.

Queue

Collection of diff. elements in FIFO (first in first out) manner.

Tree

Collection of different elements in hierarchical structure.



Application of DS in your life

Play list is an example of linked list.

File directory is the example of tree.

Stack of books.

Standing in a line.

Data structure is also called as Abstract data type or abstract data structure.

Types of ds

(i) Linear
(ii) Non-Linear
(iii) Homeogeneous
(iv) Non-homogeneous
(v) Static
(vi) Dynamic

(i) Linear

All the data items are stores or arranged in linear sequence.

Ex- Array, linked list, stack, queue etc.

(ii) Non-Linear

Items are not stored in a linear sequence.

Ex- tree & graph.

(iii) Homogeneous

All the items are of same type.

Ex- Array.

(iv) Non-homogeneous

Collection of different types of elements.

Ex- Element.

(v) Static

The size of the data structure is fixed.

Ex- array

(vi) Dynamic

The size of the ds vary.

Ex - linked list

D-13/03/20

How to determine the efficiency of the algorithm of a prog.

→ A prog. it is efficient if it take less time & less memory to execute.

(i) Time complexity
(ii) Space complexity

(i) Determines the total time needed for execution of a prog.

(ii) Determines the total space of memory for execution of a prog.

big o (O)

It is use for computing the maxm time taken by the prog. for its execution. [O()]

```
psvm( )
{
    int a=10, b=20, c;      — 1
    c = a+b;                — 1
    sopln(c);               — 1
```

O(3) & 12 byte memory (4×3)

* `for(i=0; i<n; i++)`
  `sopln(i);`

~~1+n+1+n~~
1 + n+1 + n + n
= 3n+2

O(n) [bcz it depends on n]

* `for(i=0; i<5; i++)`
  `sopln(i)`

1 + (5+1) + 5 + 5 → print
↙ initialization ↓ Condn ↓ incr.

O(5)

* `for(i=0; i<n; i++)`
  `for(j=0; j<n; j++)`
  `sopln(j)`

$O(n^2)$

*
```
for(i=0; i<n ; i++)          → n
    sopln(i);
for (i=0; i<n; i++)
    for (j=0 , j<n; j++)     → n²      m
    sopln (j) ;                n+n²
```
O(n²) bcz n² is bigger

*
```
sum=0;                  —— 1
for (i=0 ; i<n ; i++)
    sum = sum+i  —— n
sopln (sum) ;  —— 1
```
n+2 = O(n)

*
```
Psvm( )
{
  int a[]={1,2,3};          —— 1
  for (i=0 ; i<n ; i++)     } n
    if(a[i]==b)             }
    sopln(i);               —— 1
}
```
O(n)

→ Recursive fibonacci series always state $O(2^n)$ times.

best ← O(1)

avg
- O(n) → linear algorithm
- O(n log n) → super linear algorithm.
- O(log n) → log algorithm.

worst
- $O(n^c)$ → polynomial algorithm
- O(n!) → factorial algorithm

*
```
for(i=0; i<n/2; i++)
    sopln(i);
for (i=0; i<n; i++)          O(n)
    sopln(i)
```